# Introduction to Modelling

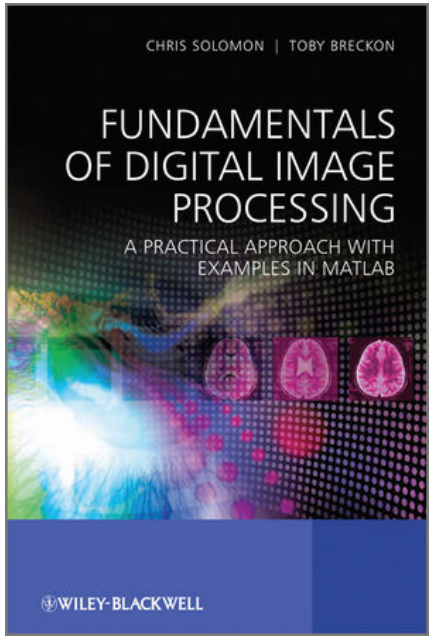
# 6. Image Processing in MATLAB

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://github.com/JimDuggan/CT248

# Overview

- Introduction to image processing with MATLAB (matrices and arrays)
- Image types
  - Binary
  - Grayscale
  - RGB
- Transformations on images

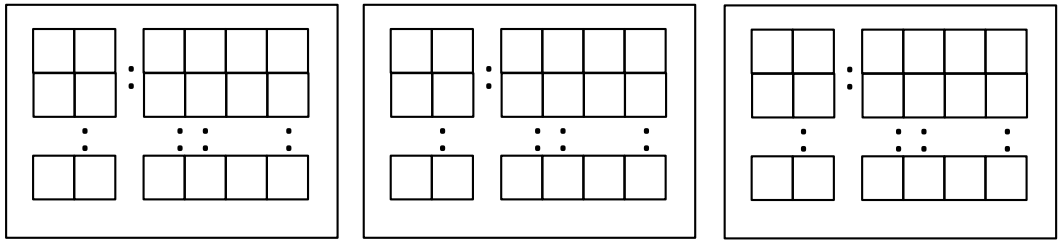



*"A digital image can be considered as a discrete representation of data possessing both spatial (layout) and intensity (colour) information"*

# Image Layout (Solomon & Breckon)

- The 2-D discrete digital image ***I(m,n)*** represents the response of some sensor (or a value of interest) at a series of fixed points *(m=1,2,..M; n=1,2,..N).*
- The indices m and n represent the rows and columns of the image.
- Individual picture elements(pixels) are referred to by their 2-D index (m,n).

```
>> whos D
  Name        Size                Bytes  Class    Attributes

  D           135x198x3           80190  uint8
```



# Image Colour

- An image contains one of more colour channels that define the intensity or colour at a particular pixel location *I(m,n)*
- In the simplest case, each pixel location only contains a single numerical value representing the signal level at that point in the image.
- Most common colour map is greyscale.
- Max number is 2^8-1 = 255

```
>> whos I
  Name        Size              Bytes  Class    Attributes

  I         256x256             65536  uint8

>> I(1:3,1:3)
3×3 uint8 matrix

   156   159   158
   160   154   157
   156   159   158
```

# Image Types

| Image Type | Description |
|---|---|
| Binary | 2-D arrays that are assign one numerical value from the set {0,1} to each pixel in the image. Often called logical images. Black corresponds to zero, white to a one. Can be represented as a simple bit stream. |
| Intensity/ Greyscale | 2-D arrays that assign one numerical value to each pixel which is representative of the intensity at this point. Range is bounded by the bit resolution of the image. |
| RGB/true-colour | 3-D arrays that assign three numerical values to each pixel, each value corresponding to the red, green and blue (RGB) image channel. Pixels accessed in MATLAB by *I(Row,Column,channel)* |
| Floating point | Do not store integer colour values. A floating point number (within a defined range) represents the intensity. Commonly stored in TIFF format, and often used to represent a scientific or medical image. |

# Key MATLAB Functions

- **imread**(filename)
  - Reads an image from a graphics file, and converts to a MATLAB array object
  - https://uk.mathworks.com/help/matlab/ref/imread.html
- **imshow**(object)
  - Displays an image
  - https://uk.mathworks.com/help/images/ref/imshow.html
- Note, MATLAB's image processing toolbox contains an extensive image processing library (manipulation & feature extraction).

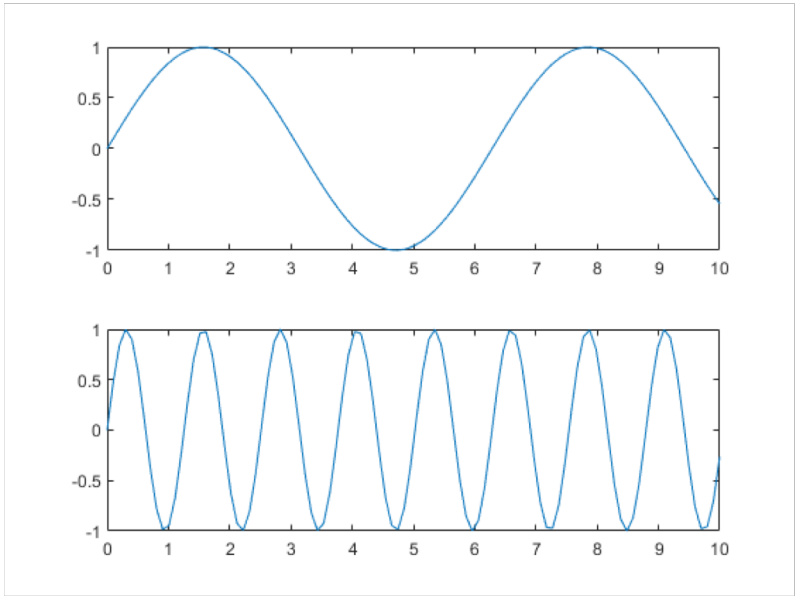
```
I=imread('cameraman.tif');

imshow(I);
```

```
>> whos I
  Name        Size              Bytes  Class    Attributes

  I         256x256             65536  uint8
```

# subplot() function

- Create axes in tiled positions
- subplot(m,n,p) divides the current figure into an m-by-n grid and creates axes in the position specified by p.
- MATLAB® numbers subplot positions by row.
- The first subplot is the first column of the first row, the second subplot is the second column of the first row, and so on.

```
subplot(2,1,1);
x = linspace(0,10);
y1 = sin(x);
plot(x,y1);

subplot(2,1,2);
y2 = sin(5*x);
plot(x,y2)
```



# Challenge 6.1

- Create a binary picture of size 60 x 100 where the values are uniformly selected from either {0,1}.

# Solution

```
data = randi([0 1],40,100);

imshow(data);
```

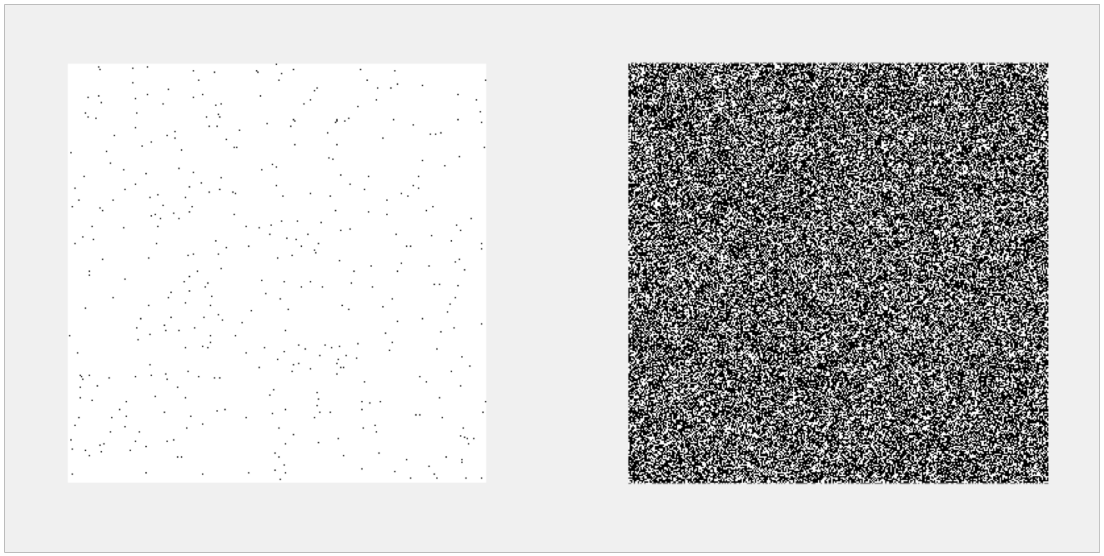
```
>> data(1:10,1:10)

ans =

     0     1     0     1     0     0     1     0     0     1
     0     0     0     1     1     1     0     1     0     0
     1     1     0     1     0     0     0     0     0     1
     0     0     1     0     1     0     0     1     0     1
     1     1     1     0     0     0     0     0     1     1
     1     1     0     1     1     0     0     0     0     0
     1     0     0     1     1     1     0     0     1     0
     0     0     0     0     1     1     1     0     0     0
     0     0     1     1     1     0     0     1     1     0
     1     1     0     0     1     0     0     0     0     0
```

# Challenge 6.2

- Create a greyscale picture of size 60 x 100 where the values are uniformly selected from either {0,255}.
- Transform any value <= 150 to black.

# Solution

```
data = randi([0 255],1000,1000);

t_data  = (data > 150) .* data;

subplot(2,2,1); imshow(data);
subplot(2,2,2); imshow(t_data);
```



```
>> data(1:5,1:5)
ans =

  210   177   206   150   231
  197    89   241    93   231
  130    63   243   103   114
  174    34   136    68   166
    2   130    90   219    35
```

```
>> t_data(1:5,1:5)
ans =

  210   177   206     0   231
  197     0   241     0   231
    0     0   243     0     0
  174     0     0     0   166
    0     0     0   219     0
```

# RGB Example

```
D=imread('onion.png');

% extract red channel
Dred  = D(:,:,1);

 % extract green channel
Dgreen = D(:,:,2);

% extract blue channel
Dblue = D(:,:,3);
```

red

green

blue

```
subplot(2,2,1); imshow(D); axis image;
subplot(2,2,2); imshow(Dred); title('red');
subplot(2,2,3); imshow(Dgreen); title('green');
subplot(2,2,4); imshow(Dblue); title('blue');
```

```
>> whos D
  Name        Size                Bytes  Class    Attributes

  D           135x198x3           80190  uint8
```

# Challenge 6.3

- Using the previous image, display pictures where each of the channels contribution is removed.

# Solution

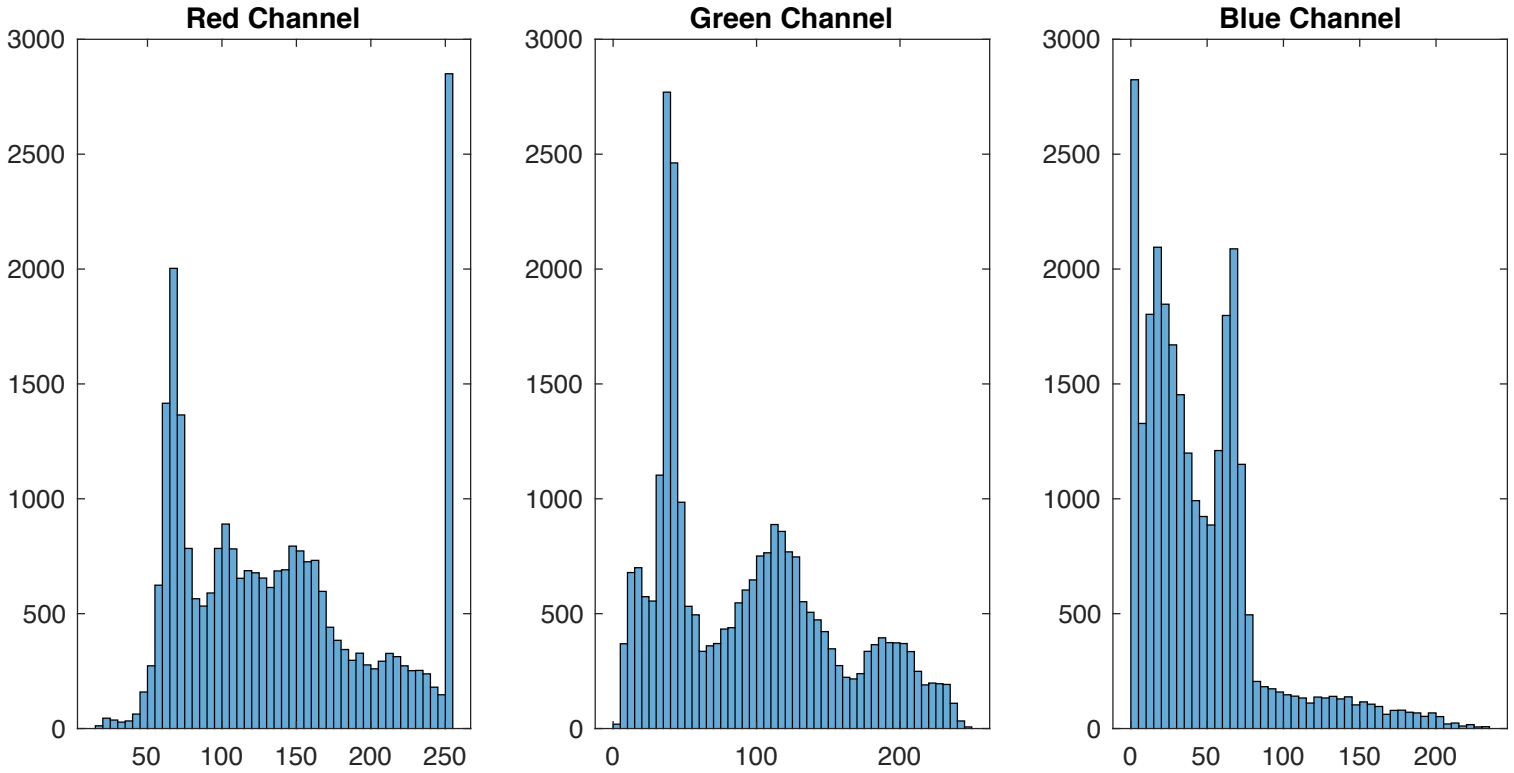
```
D=imread('onion.png');

DNoRed = D;
DNoGreen = D;
DNoBlue    = D;
```


```
DNoRed(:,:,1) = 0;
DNoGreen(:,:,2) = 0;
DNoBlue(:,:,3)    = 0;
```



```
subplot(2,2,1); imshow(D); axis image;
subplot(2,2,2); imshow(DNoRed); title('No Red');
subplot(2,2,3); imshow(DNoGreen); title('No Green');
subplot(2,2,4); imshow(DNoBlue); title('No Blue');
```

# Exploring Channel Data




```
D=imread('onion.png');
subplot(1,3,1); histogram(D(:,:,1)); title('Red Channel');
subplot(1,3,2); histogram(D(:,:,2)); title('Green Channel');
subplot(1,3,3); histogram(D(:,:,3)); title('Blue Channel');
```

# Operations on Pixels

- The most basic type of image processing is a point transform which maps the values at individual points (pixels) in the input image to corresponding points (pixels) in an output image
- In a mathematical sense, it's a one-to-one functional mapping from input to output.

- Types of operations:
  - Pixel Addition and Subtraction
  - Thresholding
  - RGB to Greyscale
  - Rotation (90$^o$)
  - Simple Cropping


# Arithmetic Operations on images

Basic arithmetic operations can be performed quickly and easily on image pixels (contrast adjustment)

```
I = imread('cameraman.tif');

O = I + 50;
O1 = I + 100;
O2 = I - 100;


subplot(2,2,1),imshow(I),title('Original Image');
subplot(2,2,2),imshow(O),title('+50');
subplot(2,2,3),imshow(O1),title('+100');
subplot(2,2,4),imshow(O2),title('-100');
```



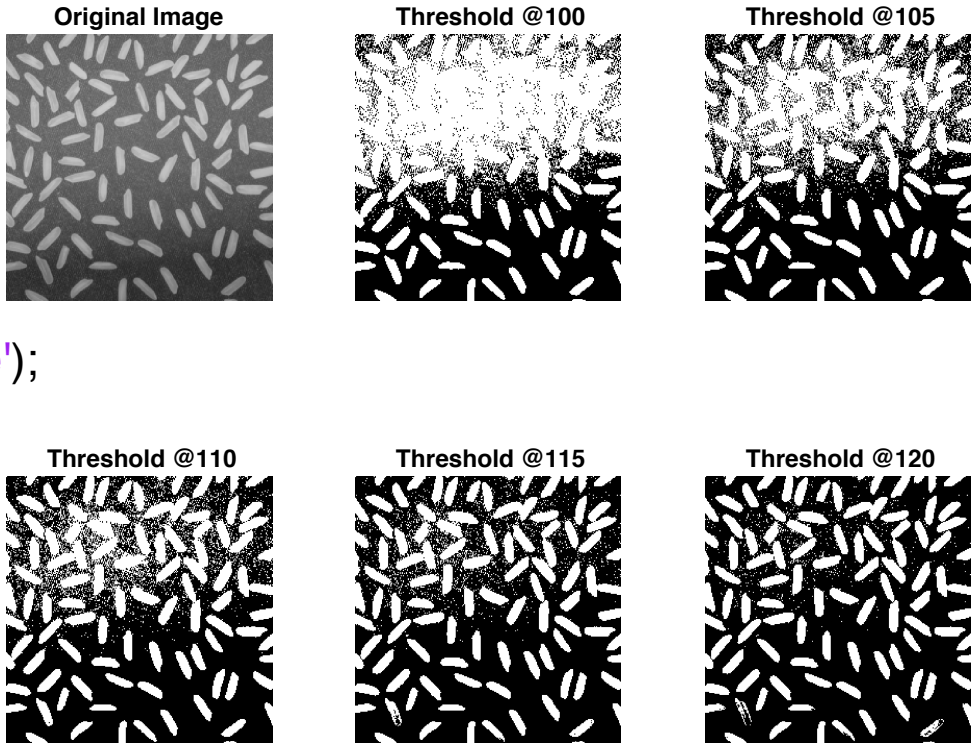

# Thresholding

- Produces a binary image from a greyscale or colour image by setting pixels to 1 or 0 depending on whether they are above or below the threshold value
- Logical operators very useful for this (TRUE = White, FALSE= Black).
- Useful to help separate the image foreground from background

```
A =

    1    2    3
    4    5    6
    7    8    9

>>
>> A > 5

ans =

  3×3 logical array

    0    0    0
    0    0    1
    1    1    1
```

# Threshold Example

```
I = imread('rice.png');

T1 = I > 100;
T2 = I > 105;
T3 = I > 110;
T4 = I > 115;
T5 = I > 120;

subplot(2,3,1),imshow(I),title('Original Image');
subplot(2,3,2),imshow(T1),title('Threshold
@100');
subplot(2,3,3),imshow(T2),title('Threshold
@105');
subplot(2,3,4),imshow(T3),title('Threshold
@110');
subplot(2,3,5),imshow(T4),title('Threshold
@115');
subplot(2,3,6),imshow(T5),title('Threshold
@120');
```



# MATLAB IP Toolbox

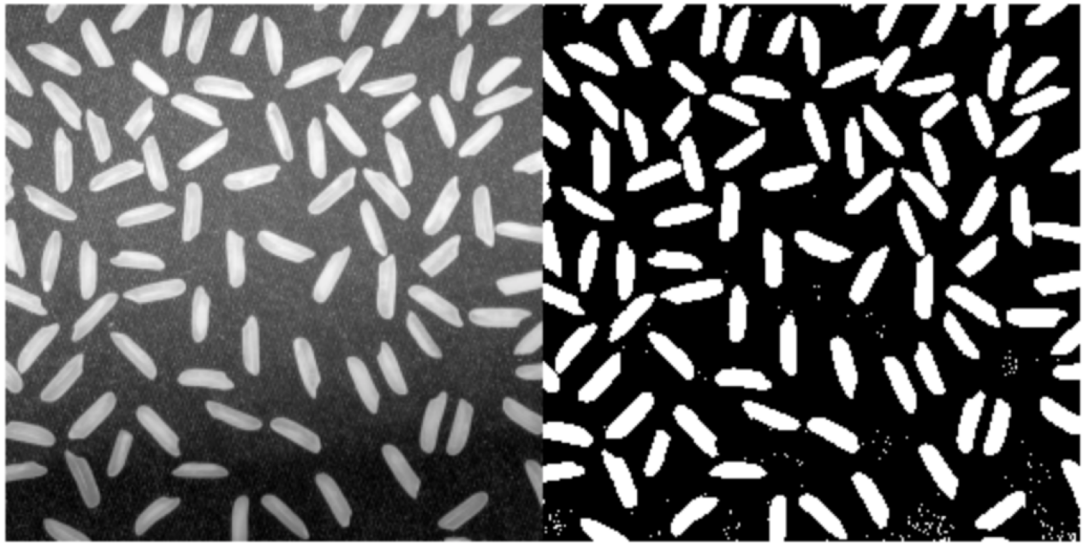
```
I = imread('rice.png');
```

Convert grayscale image to binary image.

```
BW = imbinarize(I, 'adaptive');
```

Display original image along side binary version.

```
figure
imshowpair(I,BW,'montage')
```



# RBG to Grey-scale

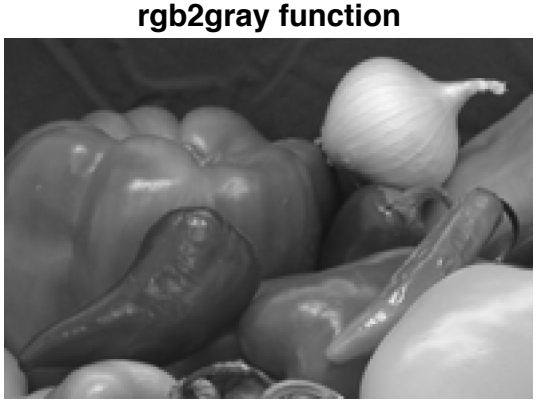
- We can convert RGB to grey-scale using a simple transform
- This conversion is the initial step in many image processing algorithms, as it simplifies (reduces) the amount of information in the image
- Feature-related information is retained, such as edges, regions, blobs, junctions.
- An RGB colour image $I_{colour}$ is converted to grey scale $I_{grey\text{-}scale}$ using the following transformation

$$I_{GS}(n,m) = \alpha\, I_c(n,m,r) + \beta\, I_c(n,m,g) + \gamma\, I_c(n,m,b)$$

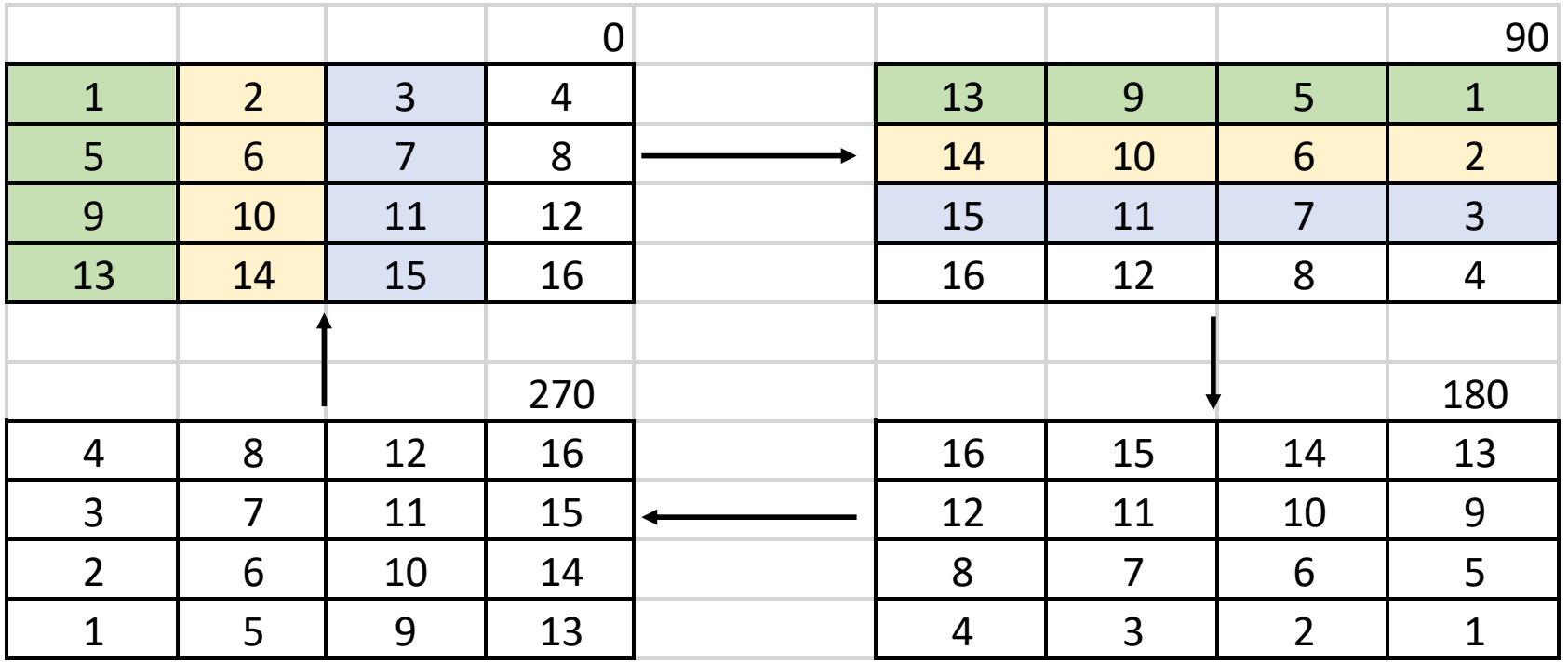
$$\alpha = 0.2989, \beta = 0.5870, \gamma = 0.1140\ (NTSC\ Standard)$$

# RBG to Grey-scale

```
D=imread('onion.png');

alpha = 0.2989;
beta = 0.5870;
gamma = 0.1140;

GS = rgb2gray(D);

GS1 = alpha * D(:,:,1) + beta * D(:,:,2) +
gamma * D(:,:,3);

subplot(2,2,1); imshow(GS); title('rgb2gray
function');
% display and label
subplot(2,2,3); imshow(GS1);
title('Transform 0.2989 0.587 0.11');
```




# Rotating a Matrix 90°




| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

| | | | |
|---|---|---|---|
| 13 | 9 | 5 | 1 |
| 14 | 10 | 6 | 2 |
| 15 | 11 | 7 | 3 |
| 16 | 12 | 8 | 4 |

| | | | |
|---|---|---|---|
| 4 | 8 | 12 | 16 |
| 3 | 7 | 11 | 15 |
| 2 | 6 | 10 | 14 |
| 1 | 5 | 9 | 13 |

| | | | |
|---|---|---|---|
| 16 | 15 | 14 | 13 |
| 12 | 11 | 10 | 9 |
| 8 | 7 | 6 | 5 |
| 4 | 3 | 2 | 1 |

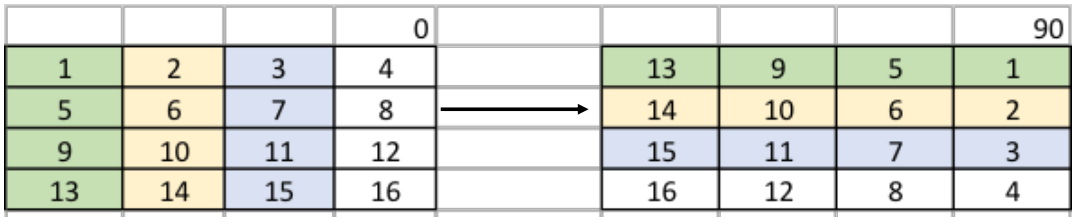
O(1,:) = fliplr ( I (:,1)' );

# Useful MATLAB function

- B = fliplr(A) returns A with its columns flipped in the left-right direction (that is, about a vertical axis).
- If A is a row vector, then fliplr(A) returns a vector of the same length with the order of its elements reversed.

```
I =
    1    2    3    4
    5    6    7    8
    9   10   11   12
   13   14   15   16

>> fliplr(I(1,:))

    4    3    2    1

>> fliplr(I(:,1))

    1
    5
    9
   13
```

# MATLAB Code

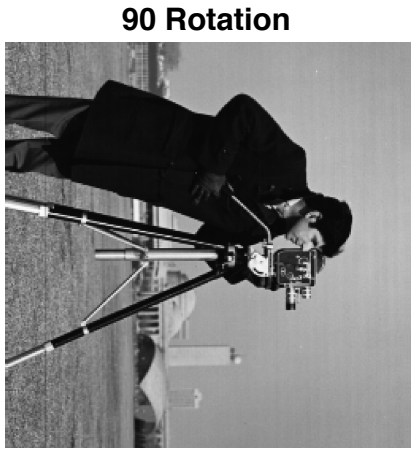



```
I = [1   2   3    4; ...
     5   6   7    8; ...
     9  10 11 12;...
    13 14 15 16]

[m, n] = size(I);

O = uint8(zeros(n,m));

for i = 1: m
    O(i,:) = fliplr(I(:,i)');
end
```

```
I =

     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16

O =

    13     9     5     1
    14    10     6     2
    15    11     7     3
    16    12     8     4
```

# Image Example

```
I = imread('cameraman.tif');

[m, n] = size(I);

O_90 = uint8(zeros(n,m));

for i = 1: m
    O_90(i,:) = fliplr(I(:,i)');
end
```


```
subplot(2,1,1),imshow(I),title('Original Image');
subplot(2,1,2),imshow(O_90),title('90 Rotation');
```

# Challenge 6.4

- Implement the 90$^{o}$ rotation in a function. Compare the results (matrix subtraction) to the MATLAB function rot90().

# Cropping an Image from the centre – Simple method

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 |

- Decide on size
- Find array centre
- Find starting rows and columns
- Filter array

# Code

```
clear;
D=imread('onion.png');

d_row = 70;
d_col =  100;

[r,c,d] = size(D);

cen_row = ceil(r/2);
cen_col  = ceil(c/2);

start_row = cen_row - (d_row/2);
start_col  = cen_col - (d_col/2);

O = D(start_row:(start_row+d_row - 1), ...
        start_col:(start_col+d_col - 1), ...
        :);

subplot(2,1,1); imshow(D); axis image;  title('135x198');

subplot(2,1,2); imshow(O); title('70x100');
```

135x198

70x100

# Challenge 6.5

- Implement the crop algorithm in a function.
- Test it for boundary conditions.

# Summary

- Introduction to Image processing
- 3 Image types (binary, greyscale, RGB)
- MATLAB array processing very powerful
- Further topics:
  - Enhancement
  - Frequency domain processing
  - Image restoration
  - Geometry
  - Morphological Processing
  - Features
  - Image Segmentation
  - Classification